ECAI 2020
G.D. Giacomo et al. (Eds.)

doi:10.3233/FAIA200113

Robust Constraint Acquisition by Sequential Analysis

Steven Prestwich[1]

Abstract.

Modeling a combinatorial problem is a hard and error-prone task requiring expertise. Constraint acquisition methods can automate this process by learning constraints from examples of solutions and (usually) non-solutions. We describe a new statistical approach based on sequential analysis that is orders of magnitude faster than existing methods, and gives accurate results on popular benchmarks. It is also robust in the sense that it can learn constraints correctly even when the data contain many errors.

1 Introduction

Constraint Programming is a powerful approach to modelling and solving decision and optimisation problems. It draws on techniques from Artificial Intelligence, Operations Research, graph theory and other areas to provide a wide range of variable types, constraints, filtering algorithms, search strategies and specification languages. A *constraint satisfaction problem* (CSP) has a set of problem variables, each with a domain of possible values, and a set or network of constraints imposed on subsets of the variables. A constraint is a relationship that must be satisfied by any solution.

However, modelling an application as a CSP, possibly with an objective function, remains a task for experts [20]. This problem, and the successes of Machine Learning at automating a wide variety of tasks, has inspired the field of *Constraint Acquisition* (CA) [1, 2, 4, 5, 16, 21, 27, 30], closely related to *Constraint Learning* [25], *Constraint Synthesis* [22] and *Empirical Model Learning* [18]. In CA we are given examples of solutions and non-solutions or failures (or positive and negative examples respectively) and the aim is to learn a constraint model that represents them. Beside the general goal of automated problem modelling, the model might be used as an explanation or compressed representation of the problem, to classify partial assignments, to show that a partial assignment cannot be placed in a class, to speed up the solution of future problems, or to find instances that optimise some objective. CA has been identified as an important topic [21], and recognised as progress toward the "holy grail" of computing in which a user simply states a problem and the computer proceeds to solve it without further programming [12].

The CA problem is defined in [25] as follows. We are given a space X of $\vec{x}$ *instances* (assignments to variables V); a space of possible constraints C; an unknown target constraint theory $T \subseteq C$; and a set of training instances E, in which *positive* instances E_+ satisfy T while *negative* instances E_- do not. The task is to find a constraint theory $H \subseteq C$ such that the positive instances in E satisfy all constraints, while the negative instances violate at least one constraint. A more detailed formal definition and theoretical results are given in [5]. *Active* methods are guided by interaction with a user or other oracle, while *passive* CA methods learn automatically. Several CA systems have been devised (see Section 5), based on version space learning [19], inductive logic programming [23, 24] and other methods, with a recent survey given in [25]. They usually require a set of *candidate constraints*, also called a *bias*, that may or may not occur in the model we are trying to learn.

An alternative approach to CA is to train a classifier to distinguish between solutions and non-solutions, then derive a constraint model from the trained classifier. This has been done for classifiers based on neural networks and decision trees [9, 18] and Naive Bayes [10]. In this paper we develop a fast method based on sequential analysis (sequential hypothesis testing). We also show that it can accurately learn constraints from noisy data in which many instances have been misclassified.

The paper is organised as follows. Section 2 describes the new method. Section 3 presents empirical results showing its accuracy and speed. Section 4 tests its robustness on noisy data. Section 5 discusses related work. Section 6 concludes the paper.

2 The method

In this section we describe the new method and discuss its properties.

2.1 A test for constraints

The key property of a constraint is that it cannot be violated by a solution, though it might be violated by a non-solution. In contrast, a non-constraint candidate might be violated by instances from both E_+ and E_-. So a simple way of checking whether a candidate $c \in C$ is a constraint is to find all training instances that violate c, and check that they are all in E_-.

However, it is not always necessary to check all instances. An obvious exception is: on encountering a c violation from E_+ we can immediately reject c as a constraint (assuming all instances are correctly classified). A more interesting question is: can we stop checking c violations after encountering a sufficient number of E_- cases? For example, if we encounter 100 violations from E_- in a row, is this enough evidence to conclude that we will *never* see a violation from E_+, and that c can reasonably be assumed to be a constraint?

There is reason to believe that this can be reliable. We have observed in experiments that a non-constraint is approximately as likely to be violated by an instance from E_+ as from E_-. As an example consider the following vertex colouring problem. The graph has 3 vertices and 3 colours, corresponding to a CSP with variables x, y, z each with domains $\{R, G, B\}$, with edges $x - y$ and $y - z$. Let the bias be the set of all possible disequalities (2 constraints $x \neq y$ and $y \neq z$, 1 non-constraint $x \neq z$) and let the training data contain all 27 possible assignments as instances. The solutions are:

[1] University College Cork, Ireland, email: s.prestwich@cs.ucc.ie

RGR RGB RBR RBG GRG GRB GBR GBG GRG GRB GBR GBG

and the non-solutions are:

RRR RRG RRB RGG RBB GRR GGR GGG GGB GBB GRR GGR GGG GGB GBB

For the constraint candidate $x \neq y$ the violating non-solutions are:

RRR RRG RRB GGR GGG GGB GGR GGG GGB

and there are no violating solutions. On the other hand, for the non-constraint candidate $x \neq z$ the violating non-solutions are:

RRR GGG

and the violating solutions are:

RGR RBR GRG GBG GRG GBG

If we randomly sample violations of a candidate, we are likely to quickly detect a solution if and only if the candidate is a non-constraint.

In practice we usually find a roughly even split for a non-constraint candidate c: in a balanced dataset any instance is in E_+ or E_- with equal probability, and if c is not a constraint then the same is true of the subset of instances that violate c. It is possible to construct cases for which this is untrue, but so far we found only cases in which c contains most or all of the problem variables. Our method would not be applied to such cases because bias size increases exponentially with candidate arity.

Assuming our observation holds, given a balanced dataset the probability of encountering a series of 100 violations from E_- in a row is extremely small (approximately 10^{-30}). It therefore seems reasonably safe to stop checking c violations after encountering 100 from E_- and none from E_+: c is almost certainly a constraint. This is the main idea explored in this paper, and we shall show that it leads to a very fast CA method that gives accurate results on standard benchmarks. First we recast the above idea as an application of *sequential analysis*.

2.2 Sequential analysis

Sequential analysis [32] is a form of hypothesis testing in which a *stopping rule* is used to stop sampling as soon as the accumulated evidence is sufficient to accept or reject the hypothesis. This has obvious benefits for patients in clinical trials [3], which can be halted as soon as it becomes obvious that an experimental treatment is harmful, or that one treatment is much more successful than another. Another application is in manufacturing, where product lots are tested for defects: lots should be accepted or rejected after as few tests as possible, to save time and costs [31]. A similar approach called Banburismus was developed independently by Turing to speed up decryption [14]. There are many more applications in the literature.

Our test can be viewed as an application of sequential analysis, in which we look for evidence to accept or reject the hypothesis that a candidate has only non-solution violations (and is therefore a constraint). When gathering evidence we can use a stopping rule to avoid sampling all violations.

2.3 SPRT

A well-known, simple and provably optimal sequential analysis algorithm is Wald's Sequential Probability Ratio Test (SPRT) [31]. Using the manufacturing application as an illustration: products are sampled and tested one by one ($m = 1, 2, \ldots$), counting the number d_m of defects found so far. If at any point $d_m < A_m$ the lot is accepted and the algorithm halts, where A_m is an *acceptance number*. On the other hand, if at any point $d_m > R_m$ the lot is rejected and the algorithm halts, where R_m is a *rejection number*. Otherwise the algorithm continues indefinitely.

A_m and R_m increase with time, as shown in Figure 1 where a lot is accepted. The formulae for computing them are

$$A_m = \frac{\ln \frac{\beta}{1-\alpha}}{\ln \frac{p_1}{p_0} + \ln \frac{1-p_1}{1-p_0}} + m \frac{\ln \frac{1-p_0}{1-p_1}}{\ln \frac{p_1}{p_0} - \ln \frac{1-p_1}{1-p_0}} \quad (1)$$

$$R_m = \frac{\ln \frac{1-\beta}{\alpha}}{\ln \frac{p_1}{p_0} + \ln \frac{1-p_1}{1-p_0}} + m \frac{\ln \frac{1-p_0}{1-p_1}}{\ln \frac{p_1}{p_0} - \ln \frac{1-p_1}{1-p_0}} \quad (2)$$

where p_0 is the defect probability below which we prefer acceptance, p_1 is the defect probability above which we prefer rejection, α controls the type I and β the type II error rate. These four parameters specify the sampling plan.

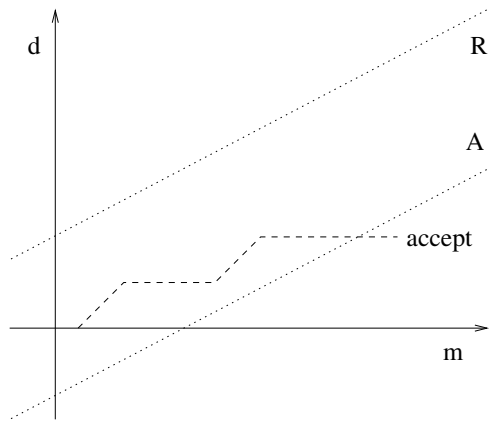

Figure 1. Illustration of SPRT

2.4 SPRT-based constraint acquisition

We now describe the new SEQACQ method (SEQuential analysis-based constraint ACQuisition) for CA. It has only two easy-to-understand parameters (A, R), one of which (R) can be set to 1 if we expect no errors in the data.

Pseudocode for the method is shown in Figure 2. For each candidate c we test whether it is violated by each of a random sequence of instances. On observing some number R of violating solutions, we reject c as a constraint. On observing some number A of violations without having rejected c, we accept it as a constraint. If we do not expect any errors in the data then we set $R = 1$, which will be the default unless stated otherwise. If neither threshold is reached before the instances are exhausted SEQACQ rejects the candidate (but see a modification in Section 2.6).

Figure 3 shows an example in which a candidate c is violated by two sampled solutions. These cause two diagonal moves (horizontal moves correspond to non-solution violations) which lead to rejection because $R = 2$. But if A non-solution violations were observed first then c would be accepted as a constraint.

```
SEQACQ(R,A)
  for each candidate c in the bias
    r ← 0   a ← 0
    repeat
      randomly choose an instance e without replacement
        (if impossible then reject c as inconclusive)
      if c is violated by e
        if the instance is a solution
          r ← r + 1
          if r ≥ R reject c as a constraint
        a ← a + 1
        if a ≥ A accept c as a constraint
```

Figure 2. The SEQACQ algorithm

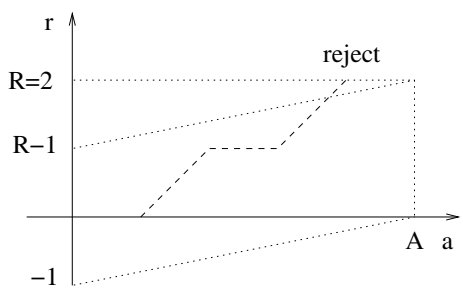

Figure 3. SEQACQ as SPRT

2.5 Algorithm parameters

SPRT has four parameters $(p_0, p_1, \alpha, \beta)$ while SEQACQ has only two (A, R). We can prove that SEQACQ is an instance of SPRT by showing that any reasonable choice of A, R $(1 \leq R < A)$ corresponds to at least one meaningful choice of SPRT parameters: *meaningful* here means that all the parameters must be probabilities and that p_1 must be greater than p_0: $0 < p_0 < p_1 < 1$ and $0 < \alpha, \beta < 1$. Using equations (1,2), in Figure 3 the rejection line at $a = 0$ has value:

$$R - 1 = \frac{\ln \frac{1-\beta}{\alpha}}{\ln \frac{p_1}{p_0} + \ln \frac{1-p_1}{1-p_0}} \tag{3}$$

the acceptance line at $a = 0$ has value:

$$-1 = \frac{\ln \frac{\beta}{1-\alpha}}{\ln \frac{p_1}{p_0} + \ln \frac{1-p_1}{1-p_0}} \tag{4}$$

and the gradient of both lines is:

$$\frac{1}{A} = \frac{\ln \frac{1-p_0}{1-p_1}}{\ln \frac{p_1}{p_0} - \ln \frac{1-p_1}{1-p_0}} \tag{5}$$

Arbitrarily setting $p_1 = ep_0$ (to satisfy $p_0 < p_1$) equation (5) becomes

$$\frac{1}{A} = \frac{\ln \frac{1-p_0}{1-ep_0}}{1 + \ln \frac{1-p_0}{1-ep_0}}$$

hence

$$p_0 = \frac{1 - e^{\frac{1}{A-1}}}{1 - e^{\frac{A}{A-1}}}$$

If $A > 1$ (required above) then $0 < p_0 < p_1 < 0.73$ so p_0, p_1 are valid probabilities. Next, subtracting equation (4) from equation (3):

$$R = \frac{\ln \frac{1-\beta}{\alpha} - \ln \frac{\beta}{1-\alpha}}{1 + \ln \frac{1-ep_0}{1-p_0}}$$

or

$$\ln \frac{1-\beta}{\alpha} - \ln \frac{\beta}{1-\alpha} = X$$

where

$$X = R\left(1 + \ln \frac{1-ep_0}{1-p_0}\right)$$

Hence

$$\alpha = \frac{1-\beta}{1-\beta+e^X\beta}$$

Now $e^X > 0$ for any X, and any expression $Y/(Y+Z)$ is in $(0,1)$ if $Y, Z > 0$, so $0 < \alpha < 1$ for any β and p_0. Hence we can choose any probability β and obtain a valid probability α, so SEQACQ is a special case of SPRT.

Note that although SPRT has 4 parameters it only has 3 degrees of freedom: one constant for each of the two lines plus a common gradient (as shown in Figure 1). However, SEQACQ has only 2 degrees of freedom because A determines both the gradient $1/A$ and the intercept $(A, 0)$. The motivation for this was to obtain a parameterisation that we believe is more intuitive for constraint programmers, but objections can be raised:

- It might be argued that by setting a very shallow gradient we are effectively using *fixed* bounds A and R, instead of *increasing* bounds as in SPRT, so SEQACQ is no longer SPRT in spirit. However, we showed above that it is technically SPRT because

our parameter values correspond to valid SPRT parameter values. Moreover, some formulations of SPRT use fixed bounds: Wald's original paper started with fixed bounds then introduced increasing bounds for computational reasons.
- Allowing fewer degrees of freedom seems likely to prevent SEQACQ from expressing the *optimal* SPRT parameters, but we shall show that it nevertheless achieves good results.
- Some researchers might prefer to use the SPRT parameterisation on the grounds that it is more statistically justifiable, or more intuitive for statisticians.

For any or all of these reasons one could instead directly apply SPRT to the CA problem: choose values for p_0, p_1, α, β and for each candidate sample its violations, testing for membership of E_- or E_+. Alternatively one could use a Bayesian hypothesis testing approach and adjust a log-likelihood ratio during sampling. However, we shall use our two parameters in this paper.

2.6 Inconclusive candidates

In experiments we found that some candidates were rejected as inconclusive when they should have been learned. This was caused by an insufficient number of violations, even on datasets of several thousand instances. It occurs with candidates that are hard to violate, for example those with high arity.

Ideally we should demand more instances to handle such cases. But to handle cases where this is impractical we instead modify SEQACQ slightly: instead of rejecting all inconclusive candidates, we accept those for which $r = 0$ and $a > 0$ and reject others. SPRT is often modified to handle inconclusive cases, yielding a Truncated SPRT (see [26] for example) that accepts or rejects them on the basis of a limited number of samples. This modification helps on some examples, but it also risks learning non-constraints. In general we prefer to obtain more instances when inconclusive cases occur.

2.7 Datasets

Different CA methods require datasets with different characteristics, for example ModelSeeker [4] only needs a few solutions, while most methods require many solutions and non-solutions. In experiments we found it necessary to create datasets with at least a thousand instances, and SEQACQ works best on datasets that are large and *balanced* (or nearly so): they have a similar number of solutions and non-solutions. Large datasets are not unusual: for example QUACQ used more than 9000 for Sudoku [2].

If the dataset has many instances of both types but is imbalanced, SEQACQ can compensate by rescaling A by a factor $|E_-|/|E_+|$: if there are more solutions then it can use a smaller value of A, while if it has more non-solutions then it should use a greater value of A to be sure of testing a significant number of violations. However, if the dataset has few non-solutions then few non-solution violations will be observed, and SEQACQ will learn few constraints (though the modification in Section 2.6 helps). And if the dataset has few solutions then SEQACQ will accept many candidates that should not be learned, because they will not be observed to be violated by any solutions.

2.8 Discussion

A reasonable question is: why not simply check whether a candidate is satisfied in all solutions? This is essentially how Valiant's SAT method [29] works (modulo algorithmic details): a candidate that is not violated by any solution is learned as a constraint. In contrast SEQACQ also requires evidence that a candidate can be violated by non-solutions. This requirement might seem odd because it is not part of the definition of a constraint, which is a relation that must satisfy all solutions.

However, the requirement has a useful consequence. Suppose we check only that candidates are not violated in any solution. Consider a particular candidate: the not-all-equal constraint on all problem variables. An instance only violates this constraint if all its variables take the same value, which is unlikely to occur randomly. Thus if the bias contains this constraint, it will almost always be learned whether or not it is part of the model. A user would quickly lose faith in a CA system that always learns such constraints.

3 Experiments

In this section we test SEQACQ on examples with fixed parameter values $A = 50$ and $R = 1$. We include results for the BAYESACQ CA method using parameter values recommended in [10] ($\alpha = 0.01$, $\kappa = 20$). Unless stated otherwise we use a bias of all possible $\{\leq, \neq, \geq\}$ constraints as in [6].

SEQACQ is implemented in the C programming language and executed on a 2.8 GHz Pentium 4 with 512 MB RAM. We shall cite run times from other papers using different machines, so they are not directly comparable to ours (except for BAYESACQ which used the same machine). However, the differences in performance we report are significantly greater than any likely difference in machine performance, as they have fairly similar clock rates: [1] used an Intel(R) Xeon(R) @ 3.40 GHz, [27] used an Intel(R) Core(TM) i5-4690K CPU @ 3.50 GHz with 8Gb of RAM, [5] used an Intel Core i7 @ 2.9 GHz with 8 Gb of RAM, and [2] used a 1.6 GHz Intel Core i5 with 4GB of RAM.

3.1 Sudoku

A Sudoku puzzle can be viewed as a CSP with an $N \times N$ array of variables, each with domain $\{1, \ldots, N\}$, and disequality constraints on pairs of variables occurring in the same rows, columns and "boxes". A 9×9 Sudoku puzzle (divided into nine 3×3 boxes) was used in [1, 2, 6, 7, 27]. Using 5000 solutions (generated by permuting a known solution) and 5000 random non-solutions, and a bias of 9720 candidates, SEQACQ learned the correct 810 disequalities in 0.05 seconds while BAYESACQ took 0.4 seconds. Both are significantly faster than other methods. Passive CONACQ took 15.6 seconds to generate background knowledge and approximately 2 seconds for acquisition [5]. In [27] MQUACA+FINDSCOPE 2 MAX_B took 85 seconds and beat five other methods. QUACQ took approximately 800 seconds and MULTIACQ approximately 900 seconds [2]. In [1] QUACQ took 2810 seconds, and a time-bounded version of QUACQ called T-QUACQ took 69 seconds.

3.2 Latin squares

A Latin square is similar to a Sudoku puzzle as a CSP, but without boxes. On a 10×10 Latin square with 5000 solutions (generated by permuting a known solution) and 5000 random non-solutions, with a bias of 2700 candidates, SEQACQ took 0.06 seconds to learn the correct 900 disequalities while BAYESACQ took 0.6 seconds. Again, both are faster than other methods. T-QUACQ took 120 seconds, compared to 7200 seconds for QUACQ [1]. In a comparison of six

CA methods in [27] the fastest was MQUACA+FINDSCOPE 2 MAX_B which took 114 seconds.

We used a larger example to further compare the two new methods: a 20×20 Latin square with a bias of 239400 candidates. SEQACQ learned the correct 7600 disequalities in 0.3 seconds while BAYESACQ took 19.3 seconds, clearly showing the advantage of early stopping.

3.3 Golomb rulers

A Golomb ruler is a set of N marks at integer positions along an imaginary ruler such that no two pairs of marks are the same distance apart. The smallest number is 0 and the largest is the ruler length L. Golomb rulers are used for CA in [1, 2, 5]. We generated 5000 solutions (by permuting a few known optimal rulers) and 5000 random non-solutions. To our usual bias we added quaternary constraint candidates $|x_i - x_j| \neq |x_{i'} - x_{j'}|$ $(i < j, i' < j', i < i', j' \neq k, k \neq k')$. The largest case usually tested is $N = 12$. SEQACQ took 0.05 seconds to correctly learn 66 disequalities and 1485 quaternary constraints, while BAYESACQ took 0.07 seconds. In contrast, QUACQ took 2257 seconds and MULTIACQ took 2335 seconds [2], while CONACQ took 2193 seconds on a smaller example ($N = 8$) [5]. In [1] QUACQ took 11972 seconds while T-QUACQ took 1184 seconds.

T-QUACQ was also tested on larger Golomb rulers and failed to converge when $N = 20$ [1]. However, for $N = 27$ with optimal length $L = 553$ both SEQACQ and BAYESACQ took 2.7 seconds to learn the 351 disequalities and 52650 quaternary constraints from a bias of 53703 candidates. The reason for the similar run times is that on this dataset all quaternary constraints are inconclusive: each quaternary constraint is unlikely to be violated by a random instance because of its high arity, so most or all of the training instances are tested.

3.4 Bandwidth vertex colouring

A benchmark used in [2, 7, 28] is the Radio Link Frequency Assignment Problem (RLFAP). The version used in [2] had 25 variables, 25 values, and a bias of 1800 candidates. QUACQ took 35 seconds, MULTIACQ took 1441 seconds, and MACQ-CO took 142 seconds. [7] used the same problem (at least the description is the same) and improved QUACQ from 1653 to 151 seconds. [28] used a larger example with 50 variables and 40 values, and a bias of 12250 candidates; four variants of QUACQ were tested, all with execution times of over 200 seconds.

We use an almost identical problem: *bandwidth colouring*, a generalisation of vertex colouring in which two adjacent vertices cannot be assigned colours that are closer in value than a specified distance. In the RLFAP these are called *interfering links* as they represent frequencies that must be different enough to prevent radio interference. (The RLFAP also has constraints forcing some adjacent vertices to have a fixed distance between them, which are called *parallel links* and are not included in bandwidth colouring. It may also have soft constraints.)

We chose one of the larger DIMACS bandwidth colouring benchmarks [15]: geometric graph GEOM100 with 100 vertices and 547 distance constraints (graph edges) with distances in the range 1–10.[2] It is known that this graph can be coloured using 50 colours, but we allow 75 colours to allow many different solutions. We generated 1000 random solutions using a local search algorithm, and 1000 random non-solutions, and used a bias of 366300 distance candidates. SEQACQ (and a newly augmented version of BAYESACQ) avoids learning redundant distance constraints by testing distances $d = m, m-1, \ldots, 2, 1$ between two vertices starting from the maximum range m which is the maximum difference between domain values, and halting on learning a constraint (greater efficiency could be achieved by performing binary search on d). SEQACQ correctly learned all the 547 constraints and their distances in 0.023 seconds while BAYESACQ took 0.24 seconds.

[2] File available at https://mat.tepper.cmu.edu/COLOR02/

3.5 Random 3-SAT

The experiments in Sections 3.1–3.4 show that SEQACQ and BAYESACQ are much faster than other methods. However, most of the benchmarks are too small for a real comparison between the two, so we now compare them on harder problems.

A propositional satisfiability (SAT) problem can be viewed as a CSP with binary domains and extensional non-binary constraints (clauses). Following [10] we generated random 3-SAT examples with different numbers V of variables, and 1000 randomly-generated instances with 5 clauses so that approximately half of the instances were solutions, and the bias is the set of all possible clauses with up to 3 literals. The results are shown in Table 1. For the largest example the bias contains over 20 million clauses. SEQACQ and BAYESACQ both learn the correct clauses, but SEQACQ is more than two orders of magnitude faster.

V	bias size (# clauses)	learning time (seconds) BAYESACQ	SEQACQ
50	1.6×10^5	1.8	0.02
100	1.3×10^6	16	0.1
150	4.5×10^6	56	0.5
200	1.1×10^7	123	0.9
250	2.1×10^7	243	1.6

Table 1. Results for 1000 random 3-SAT examples

Next we generated a 1000-variable random 3-SAT example with a bias of 1.3×10^9 clauses and 1000 instances. We also increased the size of the target to 50 clauses, obtaining an approximately balanced dataset via rejection sampling (only accepting non-solutions for the training data with probability 0.0013). SEQACQ learned the correct target in 78 seconds while BAYESSEQ took 16259 seconds. This further illustrates the improved performance of SEQACQ over BAYESACQ. It also shows that both can handle biases that are considerably larger than those used in most CA papers.

3.6 Static vs adaptive sampling

BAYESACQ is closely related to SEQACQ, but it was derived from a Naive Bayes classifier and does not use a stopping rule: it tests candidates on all training instances. It might be thought that BAYESACQ is unfairly penalised by the use of large datasets, and that it will match SEQACQ given fewer instances. We performed further experiments to test this idea.

We revisited the 20×20 Latin square problem. In tests SEQACQ used a mean of 228 instances per disequality, while BAYESACQ tests the full 10000. We used 10000 instances because the Sudoku problem needed approximately this number, but the Latin square has

fewer constraints and a smaller number of instances seems to be sufficient. In experiments BAYESACQ still learned the correct constraints with as few as 700 instances, after which it started to make errors. Using 700 BAYESACQ took 1.2 seconds, so the performance gap between the two methods can be narrowed by carefully reducing the number of instances. But SEQACQ does not need this form of tuning as it adaptively adjusts the number of instances for each candidate. Moreover, on other random samples of 700 we might find errors, so it is important to leave a safety margin by providing more instances.

We also tried the experiment on the 250-variable random 3-SAT example. BAYESACQ gave correct results with as few as 280 instances which reduced the run time to 50 seconds, but below this number it started to make errors. SEQACQ tested a mean of only 13 instances per candidate and is still more than 30 times faster, showing the benefit of adaptive sampling. Again, this difference would be greater if BAYESACQ used more instances as a safety margin.

3.7 Redundant constraints

SEQACQ is not confused by the presence of redundant constraints, unlike methods based on version spaces for which redundant constraints can prevent learning. [6] provides small examples with biases containing redundant candidates, that is some candidates in the bias are implied by others. This prevented CONACQ from eliminating some candidates, and a special technique (redundancy rules) was added to handle such cases. This necessitates the detection of higher-order redundancies. SEQACQ does not require such techniques because each candidate is tested independently, and it learned these examples correctly. However, redundant constraints should perhaps be removed from its learned model.

4 Robust constraint acquisition

To the best of our knowledge, most CA systems are not robust under errors. For systems based on version space learning, if training instances are misclassified they may become inconsistent, causing the version space to collapse. *Rough version spaces* [11] are designed to be robust but do not seem to have been applied to CA.

A statistical approach seems particularly appropriate for noisy data. The first such attempt is BAYESACQ for which it was shown that any number of errors can be overwhelmed by sufficient correct data [10]. But this is of no practical use if the errors can not be overwhelmed by correct data because the data source has a constant error rate. We now empirically test SEQACQ on data with constant error rates.

4.1 Low error rate

On the 20×20 Latin square example we deliberately misclassified 1% of the instances, and tested SEQACQ with different values of R while keeping $A = 50$ as before. The results in Table 2 show that SEQACQ is able to learn the correct constraint model for R values 4–10: no inequalities and 7600 disequalities (verified to be the correct ones). Setting R too low causes it to reject some constraints, while setting R too high causes it to mistakenly accept some candidates as constraints. Higher values of R also cause longer run times because more instances must be tested.

We repeated the experiment for the 250-variable random 3-SAT problem. The results in Table 3 shows that with R values 2–4 the correct model is learned (the 5 learned clauses were verified to be correct).

	learned constraints			time
R	$\leq$	$\geq$	$\neq$	(seconds)
1	0	0	5718	0.32
2	0	0	7360	0.34
3	0	0	7586	0.36
4	0	0	7600	0.37
5	0	0	7600	0.39
6	0	0	7600	0.43
7	0	0	7600	0.44
8	0	0	7600	0.44
9	0	0	7600	0.47
10	0	0	7600	0.50
11	0	0	7601	0.49
12	0	0	7601	0.53
13	0	0	7601	0.55
14	0	0	7601	0.56
15	0	0	7607	0.57
16	2	1	7609	0.60
17	6	1	7619	0.60
18	10	4	7630	0.62
19	19	19	7642	0.63
20	43	39	7674	0.68

Table 2. 20×20 Latin square with 1% error ($A = 50$)

	learned clauses			time
R	1	2	3	(seconds)
1	0	0	3	1.7
2	0	0	5	3.7
3	0	0	5	5.6
4	0	0	5	7.8
5	0	0	6	10.1
6	0	0	8	12.6

Table 3. Random 3-SAT with 1% error ($A = 50$)

These experiments show that SEQACQ can handle constant but low levels of misclassification in the training data, with a little parameter tuning.

4.2 High error rate

We repeated the Latin square experiment with 10% classification error. The results in Table 4 show that SEQACQ can learn the exact constraint model for a range of R-values, with A set to the larger value of 100.

	learned constraints			time
R	$\leq$	$\geq$	$\neq$	(seconds)
15	0	0	7295	0.7
20	0	0	7589	0.7
25	0	0	7600	0.9
30	0	0	7600	0.9
35	0	0	7600	1.1
40	0	0	7603	1.2
45	0	0	7629	1.3
50	12	14	7697	1.4

Table 4. 20×20 Latin square with 10% error ($A = 100$)

Similarly, on the SAT example we increased A to 200. To avoid a large number of inconclusive cases, we also increased the number of instances to 5000. The results in Table 5 show that SEQACQ learns the correct clauses over a range of R values.

	learned clauses			time
R	1	2	3	(seconds)
15	0	0	1	41
20	0	0	2	59
25	0	0	2	79
30	0	0	5	100
35	0	0	5	119
40	0	0	5	144
45	0	0	7	163
50	0	0	7	191

Table 5. Random 3-SAT with 10% error ($A = 200$)

These results show that SEQACQ can handle higher error levels, though this can require more careful parameter tuning, larger datasets and longer run times.

5 Related work

A number of CA methods are reported in the literature. ModelSeeker [4] requires only a few positive instances, and finds high-level models using global constraints. Tacle [16] learns functions and constraints from spreadsheets. CONACQ [5, 6] is based on version spaces and has passive and active versions. QUACQ [7, 8] is an active system. MULTIACQ is a related method that can learn more constraints from an example [2]. T-QUACQ [1] uses time-bounding to reduce run times. MQUACQ [27] improves QUACQ and MULTIACQ by reducing the number of generated queries and the complexity of each query. The framework of [30] learns several types of constraint model by expressing CA as a constraint problem. The Matchmaker agent [13] interacts with a user who diagnoses why an instance is not a solution. Both SEQACQ and BAYESACQ are passive. A recent trend is to use machine learning to obtain constraint and optimisation models such as neural networks, decision trees and support vector machines [9, 18, 22, 25]. BAYESACQ [10] and SEQACQ are part of this interesting new approach. Apart from these two methods we know of none that can handle noisy data.

Valiant's method [29] learns SAT formulae from instances and requires no non-solutions. It has been extended to first order logic using inductive logic programming [23, 24], which was also used by [17]. On satisfiability problems SEQACQ is related to Valiant's method in the sense that it is a generate-and-test algorithm: it generates all possible clauses of permitted length and tests each against the training data. However, there are important differences. SEQACQ has the advantage of robustness, while Valiant's method has the advantage of not requiring negative instances. Also, whereas SEQACQ tests each candidate in isolation, Valiant's algorithm first generates the set of all candidates then prunes them using each training instance in turn. This makes Valiant's method impractical when the bias is very large. A final difference is that Valiant's method will learn any clause that does not contradict the training data. In contrast SEQACQ does not learn clauses that are satisfied in all non-solutions. Hence Valiant's method learns the *most specific* model while SEQACQ is less specific. A practical advantage of this property was noted in Section 2.8.

6 Conclusion

We described a new constraint acquisition method called SEQACQ based on sequential analysis, which performs fast hypothesis testing by adaptive sampling of training instances. In experiments it accurately learns several constraint models, is orders of magnitude faster than existing methods, and is the first acquisition method to handle noisy data sources. It has only two easy-to-understand parameters, one of which has a default value that can be used if we expect no data errors. It can learn redundant constraints that cause problems for version space methods. It is also amenable to parallelisation: candidates are tested independently, so we could partition the bias into disjoint subsets and test them on a highly parallel machine such as a graphics processing unit. In future work we intend to apply it to larger problems, to use biases that include global constraints, and to use redundancy to avoid testing all candidates in the bias.

Acknowledgments

This material is based upon works supported by the Science Foundation Ireland under Grant No. 12/RC/2289-P2 which is co-funded under the European Regional Development Fund. We would also like to acknowledge the support of the Science Foundation Ireland CONFIRM Centre for Smart Manufacturing, Research Code 16/RC/3918.

REFERENCES

[1] H. A. Addi, C. Bessiere, R. Ezzahir, N. Lazaar. Time-Bounded Query Generator for Constraint Acquisition. *Proceedings of the 15th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 2018, pp. 1–17.

[2] R. Arcangioli, C. Bessiere, N. Lazaar. Multiple Constraint Acquisition. *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, 2016.

[3] J. Bartroff, T. L. Lai, M. C. Shih. Sequential Experimentation in Clinical Trials: Design and Analysis. Springer 2013.

[4] N. Beldiceanu, H. Simonis. ModelSeeker: Extracting Global Constraint Models from Positive Examples. *Data Mining and Constraint Programming, Lecture Notes in Computer Science* vol. 10101, Springer 2016, pp. 77–95.

[5] C. Bessiere, F. Koriche, N. Lazaara, B. O'Sullivan. Constraint Acquisition. *Artificial Intelligence* **244**:315–342, 2017.

[6] C. Bessiere, R. Coletta, E. C. Freuder, B. O'Sullivan. Leveraging the Learning Power of Examples in Automated Constraint Acquisition. Constraint Programming Next Challenge: Simplicity of Use. *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 3258, 2004, pp. 123–137.

[7] C. Bessiere, R. Coletta, A. Daoudi, N. Lazaar, El H. Bouyakhf. Boosting Constraint Acquisition via Generalization Queries. *Proceedings of the 21st European Conference on Artificial Intelligence*, 2014, pp. 99–104.

[8] C. Bessiere, R. Coletta, E. Hebrard, G. Katsirelos, N. Lazaar, N. Narodytska, C.-G. Quimper, T. Walsh. Constraint Acquisition via Partial Queries. *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, AAAI Press, 2013, pp. 475–481.

[9] A. Bonfietti, M. Lombardi, M. Milano. Embedding Decision Trees and Random Forests in Constraint Programming. *Proceedings of the International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science* vol. 9075, Springer 2015, pp. 74–90.

[10] D. Brown, E. C. Freuder, B. O'Sullivan, S. D. Prestwich. Constraint Acquisition Via Classification. Presented at the IJCAI'19 Workshop on Data Science Meets Optimisation, and at the CP'19 3rd Workshop on Progress Towards the Holy Grail. Submitted for journal publication.

[11] V. Dubois, M. Quafafou. Concept Learning With Approximation: Rough Version Spaces. *Rough Sets and Current Trends in Computing: Proceedings of the 3rd International Conference*, Malvern, Pennsylvania, USA, 2002, pp. 239–246.

[12] E. C. Freuder. Progress Towards the Holy Grail. *Constraints* **23**:158–171, 2018.

[13] E. C. Freuder, R. J. Wallace. Suggestion Strategies for Constraint-Based Matchmaker Agents. *International Journal on Artificial Intelligence Tools* **11**(1):3–18, 2002.

[14] I. J. Good. Turing's Anticipation of Empirical Bayes in Connection With the Cryptanalysis of the Naval Enigma. *Journal of Statistical Computation and Simulation* **66**(2):101–111, 2000.

[15] D. S. Johnson, M. A. Trick (eds.), Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* vol. 26, American Mathematical Society 1996.

[16] S. Kolb, S. Paramonov, T. Guns, L. De Raedt. Learning Constraints in Spreadsheets and Tabular Data. *Machine Learning* **106**:1441–1468, 2017.

[17] A. Lallouet, M. Lopez, L. Martin, C. Vrain. On Learning Constraint Problems. *Proceedings of the IEEE International Conference on Tools With Artificial Intelligence*, 2010, pp. 45–52.

[18] M. Lombardi, M. Milano, A. Bartolini. Empirical Decision Model Learning. *Artificial Intelligence* **244**(Supplement C):343–367, 2017.

[19] T. M. Mitchell. Machine Learning. McGraw Hill, 1997.

[20] J.-F. Puget. Constraint Programming Next Challenge: Simplicity of Use. *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 3258, 2004, pp. 5–8.

[21] B. O'Sullivan. Automated Modelling and Solving in Constraint Programming. *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, 2010, pp. 1493–1497.

[22] T. P. Pawlak, K. Krawiec. Automatic Synthesis of Constraints from Examples Using Mixed Integer Linear Programming. *European Journal of Operational Research* **261**(3):1141–1157, 2017.

[23] L. De Raedt, L. Dehaspe. Clausal Discovery. *Machine Learning* **26**:99–146, 1997.

[24] L. De Raedt, S. Džeroski. First Order *jk*-clausal Theories are PAC-Learnable. *Artificial Intelligence* **70**:375–392, 1994.

[25] L. De Raedt, A. Passerini, S. Teso. Learning Constraints from Examples. *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018, pp. 7965–7970.

[26] S. Tantara, J. B. Thomas. Truncated Sequential Probability Ratio Test. *Information Sciences* **13**(3):283–300, 1977.

[27] D. C. Tsouros, K Stergiou, P. G. Sarigiannidis. Efficient Methods for Constraint Acquisition. *Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 11008, 2018, pp, 373–388.

[28] D. C. Tsouros, K. Stergiou, C. Bessiere. Structure-Driven Multiple Constraint Acquisition. *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science vol. 11802*, 2019, pp. 709–725.

[29] L. G. Valiant. A Theory of the Learnable. *Communications of the ACM* **27**(11):1134–1142, 1984.

[30] X.-H. Vu, B. O'Sullivan. A Unifying Framework for Generalized Constraint Acquisition. *International Journal on Artificial Intelligence Tools* **17**(5):803–833, 2008.

[31] A. Wald. Sequential Tests of Statistical Hypotheses. *Ann. Math. Statist.* **16**(2):117–186, 1945.

[32] A. Wald. Sequential Analysis. John Wiley and Sons, 1947.